2nd assignment (1-2 students)

1st exercise: Full table of contents

Extend the script presented in our first JavaScript lecture to account for lower levels of headings as well (<h3>, <h4>, <h5> ...). The resulting table of contents has to be hierarchically structured, depending on the importance of every heading. Your script has to produce a table of contents similar to the one found in W3C specifications (example) and to insert it in the DOM at an appropriate position (for example, above the first <h2>).

Try to handle the problem in a generic manner instead of authoring separate code for every heading level (for instance, if in the future <h7> is added in HTML, your script shouldn't need many changes to account for that as well).

For your convenience, the script presented in the lecture is included in the zip file of the assignment with the filename **toc-h2.js**.

What to submit

- **toc.js** file

2nd exercise: CSS styled tooltips

Write a script for CSS styled tooltips (A *tooltip* is the little rectangle that a browser displays upon mouseover for elements that have the **title** attribute).

CSS doesn't yet offer a pseudo-element to allow designers to customize the appearance of tooltips. Consequently, you will have to reproduce the behavior of a tooltip with a DOM element (for example a) that will appear only when needed and will be moved as the mouse pointer moves.

Requirements:

- Use event delegation *(for performance and ease of use reasons – for example, if another script adds more DOM elements to the page, their tooltips should be taken care of, without the 2nd script "knowing" how our script works or whether it's present in the page at all)*
- Utilize the **title** attribute to take the text that should be shown in the tooltip *(for accessibility & progressive enhancement reasons)*
- The regular browser tooltip must **not** be displayed. *(otherwise, our script wouldn't only be useless, but also annoying)*

- Use only **one DOM element** for the tooltip, not a different one every time *(for performance reasons)*
- The only thing that someone has to do in order to use your script should be to include the .js file in the <head> section.
- Whether the tooltip is instantly displayed or with a delay and whether it's displayed with some kind of transition or not, is your call.

In case an ancestor and their child both have a **title** attribute, the tooltip that will be shown each time must belong to the lowest element in the hierarchy that fits (=has the mouse cursor on it). This is exactly how browsers handle this case.

Submission instructions

- The file should have the name **tooltip.js**
- The DOM element used for the tooltip must have **id="tooltip"**
- You don't have to submit a CSS file or an HTML test page, just the JavaScript code. Anything else will be ignored.

3rd exercise: Bar charts from XML data

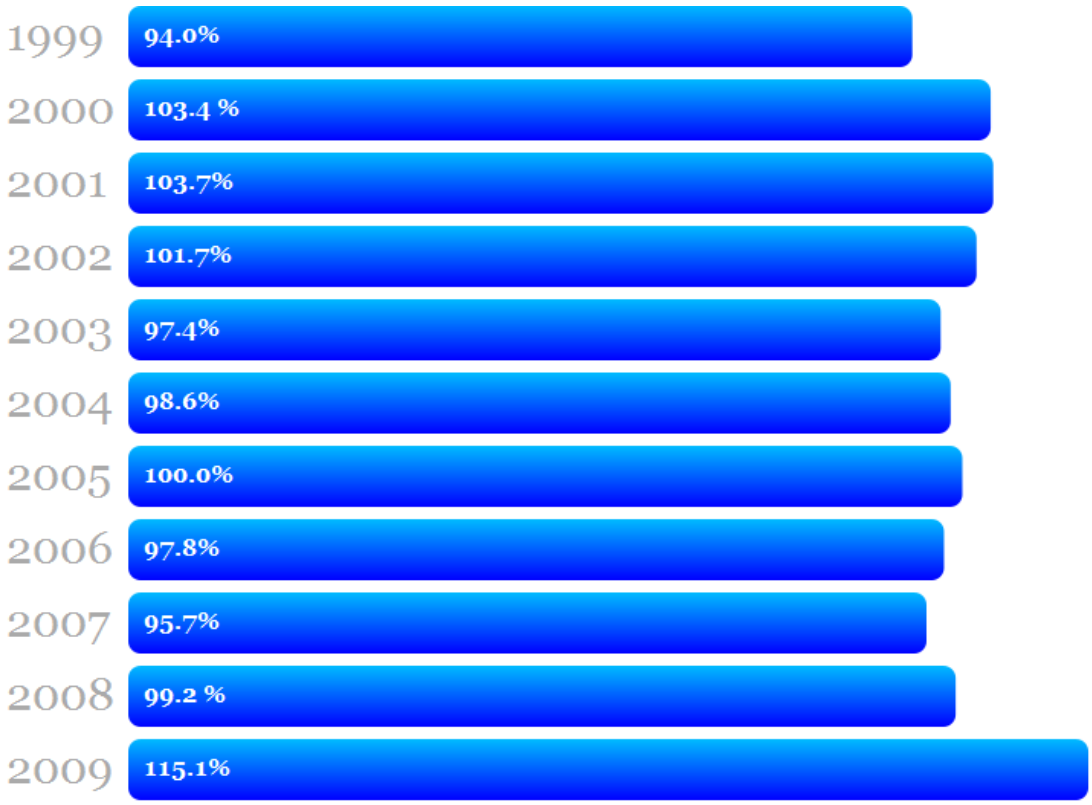
In the assignment's zip file you will find 10 XML files with data for the public debt of 10 countries during the years 1999-2009 (source: Eurostat) with a filename that follows the pattern **public-debt-[country name].xml** (ex. public-debt-greece.xml).

A: Drawing bar charts

Write an XSL Transformation that turns these data into…

1. …an SVG bar chart that presents a country's public debt per year.
2. …semantic XHTML that contains a linked CSS file that styles these data to look as a bar chart.
3. …semantic XHTML that contains a linked JavaScript file which uses the data to draw a bar chart on a **canvas** element.
4. Compare each method and list the advantages and disadvantages. Which one would you pick and why?

Example bar chart:



Try to make the 3 bar charts look similar, it will help reveal (some of) the strengths & weaknesses of each method, making it easier for you to compare them afterwards. Whether the chart will be horizontal or vertical is your call. You don't have to make it identical to the example above (for example, gradients or rounded corners are not mandatory)

Submission instructions

- Files: **transformation-svg.xsl**, **transformation-css.xsl**, **transformation-canvas.xsl**
- Files: **barchart.css** and **barchart.js** for the 2nd and 3rd method
- You don't need to submit the XML file you used for testing (and if you do, it will be ignored).
- Your comparison has to be submitted as a separate file (pdf, doc(x) or txt)

B: Using AJAX calls to fetch data & draw bar charts

Create a simple (X)HTML page that will let the user select a country and then show them the corresponding bar chart, which will be created on the client side utilizing your preferred method. The XML file needed will have to be fetched from the server via AJAX. After the first time it must be cached to avoid useless HTTP requests.

For this part of the exercise you might need XSL Transformations via JavaScript. You may read about the Firefox implementation here:

https://developer.mozilla.org/index.php?title=en/The_XSLT//JavaScript_Interface_in_Gecko

4th exercise: Form serialization & Ajax submission

Write a script that will submit a form via Ajax when the user attempts to submit it normally, using the data from the form fields and the method and action of the <form> tag. To enable this for a form, the function **ajaxForm(formObj, callback, errorCallback)** has to be called, with the parameters:

- **formObj**: A reference to the form's DOM element
- **callback**: The function that will be called if the submission was successful. It should take the XMLHttpRequest object as its first parameter and called with the form element as its context (given that its end user/developer might want to use the same function for multiple forms)
- **errorCallback**: The function that will be called in case the submission was not successful (either due to a timeout or an HTTP error). It should take the XMLHttpRequest object as its first parameter and called with the form element as its context.

To make it simpler, you don't need to worry about:

- file upload inputs
- <input type="image" />
- Which button was clicked in case more than one exist in the form (you'll serialize **any** *successful* button controls)

Submission instructions

- **ajaxforms.js** file
- You don't need to submit any HTML files that you used for testing (and if you do, they will be ignored)

5th exercise: To-do lists

Write a simple JavaScript application for TO-DO lists. Your application has to use **client-side databases** or **localStorage** (whichever you prefer) and be available **offline** as well (after being used from the web at least once, obviously). It must have the following features as a minimum:

- Add/edit/delete task
- Add/delete list

- Task reodering (not necessarily with drag & drop)
- Checkbox next to each task, to signify it as being "done".
- Search tasks

Your application must be accessible from the keyboard as well.

It's not necessary that the tasks need to be associated with a deadline, a priority or any other metadata. If, however, you decide to implement such functionality, it should **not be mandatory** to fill it in (for usability reasons).

Given that Firefox does not yet support client-side databases, if you decide to use them, your application will be tested in **Google Chrome**.

To make your application available offline, you need to read more about offline web applications here: http://www.w3.org/TR/offline-webapps/#offline (specification)

The best submission will earn a 5% bonus.

Notes

- Anything you submit will be tested in **Mozilla Firefox 3.6.x.** Alternatively, you may ask to be tested in **Google Chrome** by clearly stating it in your submission.
- You don't have to use CSS, but it's welcome, if you think it will improve the usability of your application. However, you will not lose any points if something in the CSS is wrong or invalid. Of course that does not include the CSS bar chart in 3.A.2.

Resources

This list is here to help you. Don't assume that these links are the only information you will need.

- http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf
- http://www.w3.org/TR/xslt
- http://www.w3.org/TR/html401/interact/forms.html
- http://www.w3.org/TR/SVG/
- http://dev.w3.org/html5/canvas-api/canvas-2d-api.html
- http://www.w3.org/TR/webstorage/
- http://www.w3.org/TR/webdatabase/